***C2 Server (backend) -**

- Have access to Database-MySQL
- Need pub sub service (message broker) - Redis
- Need to talk to implants (agents)
    - Functions of implants in *Implant
- Need to talk to client (API)
    - Functions of client in *Client

Making in Flask along with Gunicorn as the WSGI
Implanting

***Client (frontend) -**

- Uses API to communicate with C2
-

***Implants (agent) -**

Stager

***Tasks at hand - (priority)**

- Create c2 (using flask)
- Create database & link to the c2 (have CRUD operations)
    - Look below for database requirements (not a priority)
-

****

C2 Framework:

- The c2 server
    - Command & Control server
- The client
    - The software that allows the malware operator to communicate with the implant
- The implant
    - The malware "implanted" on a victim server
    - Can't use python for implant

****

Must use a private git repository!
Use:

- C2 server and client in python
- Backend message broker (pub/sub) with Redis
- Client UI with prompt toolkit / command prompt
- Implant in C/C++

General requirements: c2

- Handle connections to multiple operators and multiple implants

- Flask as the primary listener with a WSGI
- Postgres or MySQL as the backend database
- Use flask-SqlAlchemy to facilitate CRUD operations on agents and operators
- Using Redis to broker messages between implant, c2 and clients

For c2-> database:

- Should have several tables for data
- Implants:
    - Implant ID
    - Computer name
    - GUID
    - Integrity (authority)
    - Connecting IP address (where is it coming from)
    - Sleep
    - Session key
    - Jitter (how random of a check in is it?)
    - First seen: first check in
    - Last seen: last time you saw the agent
    - Expected check-in: based on first seen and last seen
- Commands:
- Jobs: keep track of jobs sent to implants that are in progress
    - Operator queueing a job
    - Agent pulling a job
    - Agent running a job
    - Agent response

Messaging:

- Operators should be notified whenever one of the following occurs:
    - A new implant connects to the c2 for the first time
    - A client connects to the c2
    - An operator issues a command to an agent (response)
    - An agent responds to a job that was issued by an operator
- RabbitMQ is prob easiest

Implant functionality:

- Stager: a small payload that loads the final payload from the c2 and executes it
    - Without touching disk, only in memory
    - Reflective DLL injection, PE injection, or shellcode injection
    - Authenticate with the c2, download the final payload and act as a PE loader
- Cryptography
    - Implant to c2: use asymmetric to establish a secure session to the c2
    - Symmetric: use it to encrypt all relevant configuration and strings
    - Hash function: check to see if \malware\ch0nky.txt exists, only store the hash of 'ch0nky.txt' not plaintext
- Situational awareness
    - Device a way to limit the number of implants running on the same host (mutex)

    - Read the environment variables
    - Get the Computer's MAC, IP, username & token address
- Modify config
    - C2 to communicate back to
    - Sleep time
    - Randomness to add to sleep
    - Kill date of the implant
    - Server's public key
- Execution
    - CreateProcess and redirect output to pipe (shell commands)
    - Shellcode execution (local execution, remote process execution)
- Loot
    - Programmatically loot information from the victim machine
        - Basic: chrome passwords & cookies
        - Full credit: all chromium based browsers
- Persistence
    - Some kind of persistence strategy
- File I/O
    - Read files from the victim machines and send it back to the c2, download files from the c2 and write them to disk
- Defense evasion
    - A crypter that "packs" and obfuscates your payload
    - A method for defeating AMSI
    - Anti-analysis techniques
    - Anti-sandbox techniques
    - RPC to mimic a legitimate service
    - Payload tailoring
- RPC and c2 channel
    - Implant must communicate to the C2 server using an RPC
    - Can be JSON, base64 encoded data
- *SECRET SAUCE*
    - Steganography as special thing